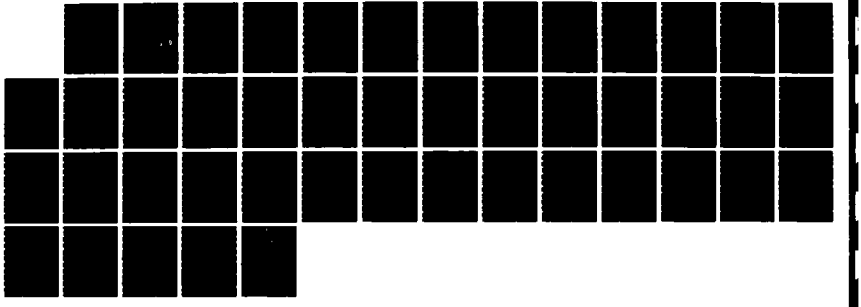
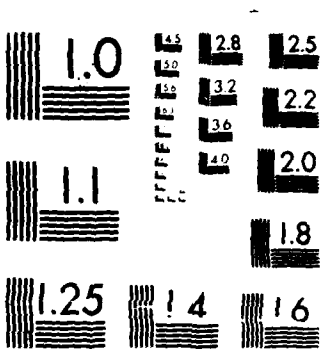
AD-A180 069 ADA (TRADENAME) COMPILER VALIDATION SUMMARY REPORT 1/1
INTERMETRICS INC INTRR (U) INFORMATION SYSTEMS AND
TECHNOLOGY CENTER W-P AFB OH ADA VALI 26 SEP 86

UNCLASSIFIED NL





MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

DTIC FILE COPY

AD-A180 069

REPORT DOCUMENTATION PAGE

1. REPORT NUMBER | 2. GOVT ACCESSION NO.	READ INSTRUCTIONS BEFORE COMPLETEING FORM 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Ada Compiler Validation Summary Report: Intermetrics Inc., Intermeterics 370/UTS Ada Compiler, Version 201.16c, IBM 3083 and IBM 4341	5. TYPE OF REPORT & PERIOD COVERED 26 SEPT 1986 to 26 SEPT 1987 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Wright-Patterson	8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION AND ADDRESS Ada Validation FAcility ASD/SIOL Wright-Patterson AFB OH 45433-6503	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Ada Joint Program Office United States Department of Defense Washington, DC 20301-3081	12. REPORT DATE 26 SEpt 1986 13. NUMBER OF PAGES 41
14. MONITORING AGENCY NAME & ADDRESS(If different from Controlling Office) Wright-Patterson	15. SECURITY CLASS (of this report) UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

DTIC ELECTE S D MAY 0 7 1987 E

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20. If different from Report)

UNCLASSIFIED

18. SUPPLEMENTARY NOTES

19. KEYWORDS (Continue on reverse side if necessary and identify by block number)

Ada Programming language, Ada Compiler Validation Summary Report, Ada Compiler Validation Capability, ACVC, Validation Testing, Ada Validation Office, AVO, Ada Validation Facility, AVF, ANSI/MIL-STD-1815A, Ada Joint Program Office, AJPO

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See Attached.

Ada® Compiler Validation Summary Report:

Compiler Name: Intermetrics 370/UTS Ada Compiler, Version 201.16c

Hosts and Targets:
- IBM 3083 under UTS, Version 2.3
- IBM 4341 under UTS, Version 2.3

Testing Completed 26 September 1986 Using ACVC 1.8

This report has been reviewed and is approved.

Georgeanne Chitwood

Ada Validation Facility
Georgeanne Chitwood
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

Accession For	
NTIS GRA&I	X
DTIC TAB	☐
Unannounced	☐
Justification	
By	
Distribution/	
Availability Codes	
Dist	Avail and/or Special
A-1	

Robert W. Lee for JFK

Ada Validation Organization
Dr. John F. Kramer
Institute for Defense Analyses
Alexandria VA

Virginia L. Castor

Ada Joint Program Office
Virginia L. Castor
Director
Department of Defense
Washington DC

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

87 5 6 123

AVF Control Number: AVF-VSR-44.1286

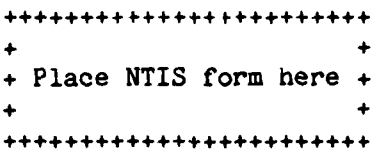
Ada® COMPILER
VALIDATION SUMMARY REPORT:
Intermetrics Inc.
Intermetrics 370/UTS Ada Compiler, Version 201.16c
IBM 3083 and IBM 4341

Completion of On-Site Testing:
26 September 1986

Prepared By:
Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

Prepared For:
Ada Joint Program Office
United States Department of Defense
Washington, D.C.

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).


Place NTIS form here

EXECUTIVE SUMMARY

This Validation Summary Report (VSR) summarizes the results and conclusions of validation testing performed on the Intermetrics 370/UTS Ada Compiler, Version 201.16c, using Version 1.8 of the Ada® Compiler Validation Capability (ACVC). The Intermetrics 370/UTS Ada Compiler is hosted on an IBM 3083 and an IBM 4341 operating under UTS, Version 2.3. Programs processed by this compiler may be executed on an IBM 3083 or an IBM 4341 operating under UTS, Version 2.3.

On-site testing was performed 22 September 1986 through 26 September 1986 at Intermetrics Inc., Cambridge MA, under the direction of the Ada Validation Facility (AVF), according to Ada Validation Organization (AVO) policies and procedures. The AVF identified 2210 of the 2399 tests in ACVC Version 1.8 to be processed during on-site testing of the compiler. The 19 tests withdrawn at the time of validation testing, as well as the 170 executable tests that make use of floating-point precision exceeding that supported by the implementation, were not processed. After the 2210 tests were processed, results for Class A, C, D, or E tests were examined for correct execution. Compilation listings for Class B tests were analyzed for correct diagnosis of syntax and semantic errors. Compilation and link results of Class L tests were analyzed for correct detection of errors. There were 29 of the processed tests determined to be inapplicable; the remaining 2181 tests were passed.

The results of validation are summarized in the following table:

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	102	251	334	235	161	97	134	262	128	32	218	227	2181
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	74	86	12	0	0	5	0	2	0	0	6	199
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

The AVF concludes that these results demonstrate acceptable conformity to ANSI/MIL-STD-1815A Ada.

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

TABLE OF CONTENTS

CHAPTER 1 INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT 1-2
1.2 USE OF THIS VALIDATION SUMMARY REPORT 1-2
1.3 REFERENCES 1-3
1.4 DEFINITION OF TERMS 1-3
1.5 ACVC TEST CLASSES 1-4

CHAPTER 2 CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED 2-1
2.2 IMPLEMENTATION CHARACTERISTICS 2-2

CHAPTER 3 TEST INFORMATION

3.1 TEST RESULTS 3-1
3.2 SUMMARY OF TEST RESULTS BY CLASS 3-1
3.3 SUMMARY OF TEST RESULTS BY CHAPTER 3-2
3.4 WITHDRAWN TESTS 3-2
3.5 INAPPLICABLE TESTS 3-2
3.6 SPLIT TESTS 3-4
3.7 ADDITIONAL TESTING INFORMATION 3-4
3.7.1 Prevalidation 3-4
3.7.2 Test Method 3-4
3.7.3 Test Site 3-5

APPENDIX A COMPLIANCE STATEMENT

APPENDIX B APPENDIX F OF THE Ada STANDARD

APPENDIX C TEST PARAMETERS

APPENDIX D WITHDRAWN TESTS

CHAPTER 1

INTRODUCTION

This Validation Summary Report (VSR) describes the extent to which a specific Ada compiler conforms to the Ada Standard, ANSI/MIL-STD-1815A. This report explains all technical terms used within it and thoroughly reports the results of testing this compiler using the Ada Compiler Validation Capability (ACVC). An Ada compiler must be implemented according to the Ada Standard, and any implementation-dependent features must conform to the requirements of the Ada Standard. The Ada Standard must be implemented in its entirety, and nothing can be implemented that is not in the Standard.

Even though all validated Ada compilers conform to the Ada Standard, it must be understood that some differences do exist between implementations. The Ada Standard permits some implementation dependencies--for example, the maximum length of identifiers or the maximum values of integer types. Other differences between compilers result from characteristics of particular operating systems, hardware, or implementation strategies. All of the dependencies observed during the process of testing this compiler are given in this report.

The information in this report is derived from the test results produced during validation testing. The validation process includes submitting a suite of standardized tests, the ACVC, as inputs to an Ada compiler and evaluating the results. The purpose of validating is to ensure conformity of the compiler to the Ada Standard by testing that the compiler properly implements legal language constructs and that it identifies and rejects illegal language constructs. The testing also identifies behavior that is implementation dependent but permitted by the Ada Standard. Six classes of tests are used. These tests are designed to perform checks at compile time, at link time, and during execution.

INTRODUCTION

1.1 PURPOSE OF THIS VALIDATION SUMMARY REPORT

This VSR documents the results of the validation testing performed on an Ada compiler. Testing was carried out for the following purposes:

- To attempt to identify any language constructs supported by the compiler that do not conform to the Ada Standard
- To attempt to identify any unsupported language constructs required by the Ada Standard
- To determine that the implementation-dependent behavior is allowed by the Ada Standard

Testing of this compiler was conducted by SofTech, Inc., under the direction of the AVF according to the procedures established by the Ada Validation Organization (AVO). On-site testing was conducted from 22 September 1986 through 26 September 1986 at Intermetrics Inc., Cambridge MA.

1.2 USE OF THIS VALIDATION SUMMARY REPORT

Consistent with the national laws of the originating country, the AVO may make full and free public disclosure of this report. In the United States, this is provided in accordance with the "Freedom of Information Act" (5 U.S.C. #552). The results of this validation apply only to the computers, operating systems, and compiler versions identified in this report.

The organizations represented on the signature page of this report do not represent or warrant that all statements set forth in this report are accurate and complete, or that the subject compiler has no nonconformities to the Ada Standard other than those presented. Copies of this report are available to the public from:

Ada Information Clearinghouse
Ada Joint Program Office
OUSDRE
The Pentagon, Rm 3D-139 (Fern Street)
Washington DC 20301-3081

or from:

Ada Validation Facility
ASD/SIOL
Wright-Patterson AFB OH 45433-6503

Questions regarding this report or the validation test results should be directed to the AVF listed above or to:

Ada Validation Organization
Institute for Defense Analyses
1801 North Beauregard Street
Alexandria VA 22311

1.3 REFERENCES

1. Reference Manual for the Ada Programming Language, ANSI/MIL-STD-1815A, FEB 1983.

2. Ada Validation Organization: Policies and Procedures, MITRE Corporation, JUN 1982, PB 83-110601.

3. Ada Compiler Validation Capability Implementers' Guide, SofTech, Inc., SEP 1986.

1.4 DEFINITION OF TERMS

ACVC	The Ada Compiler Validation Capability. A set of programs that evaluates the conformity of a compiler to the Ada language specification, ANSI/MIL-STD-1815A.
Ada Standard	ANSI/MIL-STD-1815A, February 1983.
Applicant	The agency requesting validation.
AVF	The Ada Validation Facility. In the context of this report, the AVF is responsible for conducting compiler validations according to established policies and procedures.
AVO	The Ada Validation Organization. In the context of this report, the AVO is responsible for setting procedures for compiler validations.
Compiler	A processor for the Ada language. In the context of this report, a compiler is any language processor, including cross-compilers, translators, and interpreters.
Failed test	A test for which the compiler generates a result that demonstrates nonconformity to the Ada Standard.
Host	The computer on which the compiler resides.

Inapplicable test	A test that uses features of the language that a compiler is not required to support or may legitimately support in a way other than the one expected by the test.
Passed test	A test for which a compiler generates the expected result.
Target	The computer for which a compiler generates code.
Test	A program that checks a compiler's conformity regarding a particular feature or features to the Ada Standard. In the context of this report, the term is used to designate a single test, which may comprise one or more files.
Withdrawn test	A test found to be incorrect and not used to check conformity to the Ada language specification. A test may be incorrect because it has an invalid test objective, fails to meet its test objective, or contains illegal or erroneous use of the language.

1.5 ACVC TEST CLASSES

Conformity to the Ada Standard is measured using the ACVC. The ACVC contains both legal and illegal Ada programs structured into six test classes: A, B, C, D, E, and L. The first letter of a test name identifies the class to which it belongs. Class A, C, D, and E tests are executable, and special program units are used to report their results during execution. Class B tests are expected to produce compilation errors. Class L tests are expected to produce link errors.

Class A tests check that legal Ada programs can be successfully compiled and executed. However, no checks are performed during execution to see if the test objective has been met. For example, a Class A test checks that reserved words of another language (other than those already reserved in the Ada language) are not treated as reserved words by an Ada compiler. A Class A test is passed if no errors are detected at compile time and the program executes to produce a PASSED message.

Class B tests check that a compiler detects illegal language usage. Class B tests are not executable. Each test in this class is compiled and the resulting compilation listing is examined to verify that every syntax or semantic error in the test is detected. A Class B test is passed if every illegal construct that it contains is detected by the compiler.

Class C tests check that legal Ada programs can be correctly compiled and executed. Each Class C test is self-checking and produces a PASSED, FAILED, or NOT APPLICABLE message indicating the result when it is executed.

Class D tests check the compilation and execution capacities of a compiler. Since there are no capacity requirements placed on a compiler by the Ada Standard for some parameters--for example, the number of identifiers

permitted in a compilation or the number of units in a library--a compiler may refuse to compile a Class D test and still be a conforming compiler. Therefore, if a Class D test fails to compile because the capacity of the compiler is exceeded, the test is classified as inapplicable. If a Class D test compiles successfully, it is self-checking and produces a PASSED or FAILED message during execution.

Each Class E test is self-checking and produces a NOT APPLICABLE, PASSED, or FAILED message when it is compiled and executed. However, the Ada Standard permits an implementation to reject programs containing some features addressed by Class E tests during compilation. Therefore, a Class E test is passed by a compiler if it is compiled successfully and executes to produce a PASSED message, or if it is rejected by the compiler for an allowable reason.

Class L tests check that incomplete or illegal Ada programs involving multiple, separately compiled units are detected and not allowed to execute. Class L tests are compiled separately and execution is attempted. A Class L test passes if it is rejected at link time--that is, an attempt to execute the main program must generate an error message before any declarations in the main program or any units referenced by the main program are elaborated.

Two library units, the package REPORT and the procedure CHECK_FILE, support the self-checking features of the executable tests. The package REPORT provides the mechanism by which executable tests report PASSED, FAILED, or NOT APPLICABLE results. It also provides a set of identity functions used to defeat some compiler optimizations allowed by the Ada Standard that would circumvent a test objective. The procedure CHECK_FILE is used to check the contents of text files written by some of the Class C tests for chapter 14 of the Ada Standard. The operation of these units is checked by a set of executable tests. These tests produce messages that are examined to verify that the units are operating correctly. If these units are not operating correctly, then the validation is not attempted.

The text of the tests in the ACVC follow conventions that are intended to ensure that the tests are reasonably portable without modification. For example, the tests make use of only the basic set of 55 characters, contain lines with a maximum length of 72 characters, use small numeric values, and place features that may not be supported by all implementations in separate tests. However, some tests contain values that require the test to be customized according to implementation-specific values--for example, an illegal file name. A list of the values used for this validation are listed in Appendix C.

A compiler must correctly process each of the tests in the suite and demonstrate conformity to the Ada Standard by either meeting the pass criteria given for the test or by showing that the test is inapplicable to the implementation. Any test that was determined to contain an illegal language construct or an erroneous language construct is withdrawn from the ACVC and, therefore, is not used in testing a compiler. The tests withdrawn at the time of validation are given in Appendix D.

CHAPTER 2

CONFIGURATION INFORMATION

2.1 CONFIGURATION TESTED

The candidate compilation system for this validation was tested under the following configuration:

Compiler: Intermetrics 370/UTS Ada Compiler, Version 201.16c

ACVC Version: 1.8

Certificate Expiration Date: 16 December 1987

Host and Target Computer:

Machine:	IBM 3083
Operating System:	UTS Version 2.3
Memory Size:	24 megabytes

Host and Target Computer:

Machine:	IBM 4341
Operating System:	UTS Version 2.3
Memory Size:	12 megabytes

2.2 IMPLEMENTATION CHARACTERISTICS

One of the purposes of validating compilers is to determine the behavior of a compiler in those areas of the Ada Standard that permit implementations to differ. Class D and E tests specifically check for such implementation differences. However, tests in other classes also characterize an implementation. This compiler is characterized by the following interpretations of the Ada Standard:

- Capacities.

  The compiler correctly processes tests containing loop statements nested to 17 levels and recursive procedures separately compiled as subunits nested to 17 levels. The compiler could not process block statements nested to 65 levels. It correctly processes a compilation containing 723 variables in the same declarative part. (See tests D55A03A..H (8 tests), D56001B, D64005E..G (3 tests), and D29002K.)

- Universal integer calculations.

  An implementation is allowed to reject universal integer calculations having values that exceed SYSTEM.MAX_INT. This implementation does not reject such calculations and processes them correctly. (See tests D4A002A, D4A002B, D4A004A, and D4A004B.)

- Predefined types.

  This implementation supports the additional predefined type SHORT_FLOAT in the package STANDARD. (See tests B86001C and B86001D.)

- Based literals.

  An implementation is allowed to reject a based literal with a value exceeding SYSTEM.MAX_INT during compilation, or it may raise NUMERIC_ERROR or CONSTRAINT_ERROR during execution. This implementation raises NUMERIC_ERROR during execution. (See test E24101A.)

- Array types.

  An implementation is allowed to raise NUMERIC_ERROR or CONSTRAINT_ERROR for an array having a 'LENGTH that exceeds STANDARD.INTEGER'LAST and/or SYSTEM.MAX_INT.

A packed BOOLEAN array having a 'LENGTH exceeding INTEGER'LAST raises NUMERIC_ERROR when the array type is declared. (See test C52103X.)

A packed two-dimensional BOOLEAN array with more than INTEGER'LAST components raises NUMERIC_ERROR when the array type is declared. (See test C52104Y.)

A null array with one dimension of length greater than INTEGER'LAST may raise NUMERIC_ERROR or CONSTRAINT_ERROR either when declared or assigned. Alternately, an implementation may accept the declaration. However, lengths must match in array slice assignments. This implementation raises NUMERIC_ERROR when the array type is declared. (See test E52103Y.)

In assigning one-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. In assigning two-dimensional array types, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Discriminated types.

  During compilation, an implementation is allowed to either accept or reject an incomplete type with discriminants that is used in an access type definition with a compatible discriminant constraint. This implementation accepts such subtype indications. (See test E38104A.)

  In assigning record types with discriminants, the expression does not appear to be evaluated in its entirety before CONSTRAINT_ERROR is raised when checking whether the expression's subtype is compatible with the target's subtype. (See test C52013A.)

- Aggregates.

  In the evaluation of a multi-dimensional aggregate, all choices appear to be evaluated before checking against the index type. (See tests C43207A and C43207B.)

  In the evaluation of an aggregate containing subaggregates, all choices are not evaluated before being checked for identical bounds. (See test E43212B.)

All choices are evaluated before CONSTRAINT_ERROR is raised if a bound in a nonnull range of a nonnull aggregate does not belong to an index subtype. (See test E43211B.)

. Functions.

An implementation may allow the declaration of a parameterless function and an enumeration literal having the same profile in the same immediate scope, or it may reject the function declaration. If it accepts the function declarations, the use of the enumeration literal's identifier denotes the function. This implementation rejects the declarations. (See test E66001D.)

. Representation clauses.

The Ada Standard does not require an implementation to support representation clauses. If a representation clause is not supported, then the implementation must reject it. While the operation of representation clauses is not checked by Version 1.8 of the ACVC, they are used in testing other language features. This implementation rejects 'SIZE and 'STORAGE_SIZE for tasks, 'STORAGE_SIZE for collections, and 'SMALL clauses. Enumeration representation clauses, including those that specify noncontiguous values, appear not to be supported. (See tests C55B16A, C87B62A, C87B62B, C87B62C, and BC1002A.)

. Pragmas.

The pragma INLINE is supported for procedures. The pragma INLINE is supported for functions. (See tests CA3004E and CA3004F.)

. Input/output.

The package SEQUENTIAL_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. The package DIRECT_IO cannot be instantiated with unconstrained array types and record types with discriminants without defaults. (See tests AE2101C, AE2101H, CE2201D, CE2201E, and CE2401D.)

An existing text file can be opened in OUT_FILE mode, can be created in OUT_FILE mode, and can be created in IN_FILE mode. (See test EE3102C.)

More than one internal file can be associated with each external file for text I/O for reading only or writing only. (See tests CE3111A..E (5 tests).)

More than one internal file can be associated with each external file for sequential I/O for both reading and writing. (See tests CE2107A..F (6 tests).)

More than one internal file can be associated with each external file for direct I/O for both reading and writing. (See tests CE2107A..F (6 tests).)

An external file associated with more than one internal file can be deleted. (See test CE2110B.)

Temporary sequential files are given a name. Temporary direct files are given a name. Temporary files given names are deleted when they are closed. (See tests CE2108A and CE2108C.)

. Generics.

Body and subunits of a generic unit must be in the same compilation as the specification if instantiations precede them. (See tests CA2009C and CA2009F.)

CHAPTER 3

TEST INFORMATION

3.1 TEST RESULTS

Version 1.8 of the ACVC contains 2399 tests. When validation testing of Intermetrics 370/UTS Ada Compiler was performed, 19 tests had been withdrawn. The remaining 2380 tests were potentially applicable to this validation. The AVF determined that 199 tests were inapplicable to this implementation, and that the 2181 applicable tests were passed by the implementation.

The AVF concludes that the testing results demonstrate acceptable conformity to the Ada Standard.

3.2 SUMMARY OF TEST RESULTS BY CLASS

RESULT	TEST CLASS						TOTAL
	A	B	C	D	E	L	
Passed	67	862	1181	12	13	46	2181
Failed	0	0	0	0	0	0	0
Inapplicable	2	5	187	5	0	0	199
Withdrawn	0	7	12	0	0	0	19
TOTAL	69	874	1380	17	13	46	2399

3.3 SUMMARY OF TEST RESULTS BY CHAPTER

RESULT	CHAPTER												TOTAL
	2	3	4	5	6	7	8	9	10	11	12	14	
Passed	102	251	334	235	161	97	134	262	128	32	218	227	2181
Failed	0	0	0	0	0	0	0	0	0	0	0	0	0
Inapplicable	14	74	86	12	0	0	5	0	2	0	0	6	199
Withdrawn	0	5	5	0	0	1	1	2	4	0	1	0	19
TOTAL	116	330	425	247	161	98	140	264	134	32	219	233	2399

3.4 WITHDRAWN TESTS

The following 19 tests were withdrawn from ACVC Version 1.8 at the time of this validation:

C32114A	C41404A	B74101B
B33203C	B45116A	C87B50A
C34018A	C48008A	C92005A
C35904A	B49006A	C940ACA
B37401A	B4A010C	CA3005A..D (4 tests)
		BC3204C

See Appendix D for the reason that each of these tests was withdrawn.

3.5 INAPPLICABLE TESTS

Some tests do not apply to all compilers because they make use of features that a compiler is not required by the Ada Standard to support. Others may depend on the result of another test that is either inapplicable or withdrawn. For this validation attempt, 199 tests were inapplicable for the reasons indicated:

- C34001D, B52004E, B55B09D, and C55B07B use SHORT_INTEGER which is not supported by this compiler.
- C34001E, B52004D, B55B09C, and C55B07A use LONG_INTEGER which is not supported by this compiler.
- C34001G and C35702B use LONG_FLOAT which is not supported by this compiler.

- D55A03E..H (4 tests) require 31 to 65 levels of loop nesting which is greater than this implementation supports.

- D56001B requires 65 levels of block nesting which is greater than this implementation supports.

- C55B16A makes use of an enumeration representation clause containing noncontiguous values which is not supported by this compiler.

- B86001D requires a predefined numeric type other than those defined by the Ada language in package STANDARD. There is no such type for this implementation.

- C86001F redefines package SYSTEM, but TEXT_IO is made obsolete by this new definition in this implementation and the test cannot be executed since the package REPORT is dependent on the package TEXT_IO.

- C87B62A..C (3 tests) use length clauses which are not supported by this compiler. The length clauses are rejected during compilation.

- CA2009C and CA2009F compile generic subunits in separate compilation files. For this implementation, the body and subunits of a generic unit must be in the same compilation as the specification if instantiations precede them.

- AE2101C, CE2201D, and CE2201E use an instantiation of package SEQUENTIAL_IO with unconstrained array types which is not supported by this compiler.

- AE2101H and CE2401D use an instantiation of package DIRECT_IO with unconstrained array types which is not supported by this compiler.

- CE3111B assumes that if the same external file is open for both reading and writing, then characters written may be immediately re-read, without a new-line/reset/close separating the read and write. This implementation buffers output and requires that a reset be issued between writing and reading from the same external file, if the read wants to be sure to see the effect of the write.

- The following 170 tests require a floating-point accuracy that exceeds the maximum of 15 digits supported by the implementation:

C24113L..Y (14 tests)
C35705L..Y (14 tests)
C35706L..Y (14 tests)
C35707L..Y (14 tests)
C35708L..Y (14 tests)
C35802L..Y (14 tests)
C45241L..Y (14 tests)

TEST INFORMATION

C45321L..Y (14 tests)
C45421L..Y (14 tests)
C45424L..Y (14 tests)
C45521L..Z (15 tests)
C45621L..Z (15 tests)

3.6 SPLIT TESTS

If one or more errors do not appear to have been detected in a Class B test because of compiler error recovery, then the test is split into a set of smaller tests that contain the undetected errors. These splits are then compiled and examined. The splitting process continues until all errors are detected by the compiler or until there is exactly one error per split. Any Class A, Class C, or Class E test that cannot be compiled and executed because of its size is split into a set of smaller subtests that can be processed.

Splits were required for 2 Class B tests:

BA1101C BC3205D

3.7 ADDITIONAL TESTING INFORMATION

3.7.1 Prevalidation

Prior to validation, a set of test results for ACVC Version 1.8 produced by the Intermetrics 370/UTS Ada Compiler was submitted to the AVF by the applicant for review. Analysis of these results demonstrated that the compiler successfully passed all applicable tests, and the compiler exhibited the expected behavior on all inapplicable tests.

3.7.2 Test Method

Testing of the Intermetrics 370/UTS Ada Compiler using ACVC Version 1.8 was conducted on-site by a validation team from the AVF. The configuration consisted of an IBM 3083 operating under UTS, Version 2.3. The following configuration was also tested using a subset of the ACVC:

Host:	Target:
IBM 4341	IBM 4341

A magnetic tape containing all tests except for withdrawn tests and tests requiring unsupported floating-point precisions was taken on-site by the validation team for processing. Tests that make use of

implementation-specific values were customized before being written to the magnetic tape. Tests requiring splits during the prevalidation testing were included in their split form on the magnetic tape.

The contents of the magnetic tape were loaded directly onto the host computer.

After the test files were loaded to disk, the full set of tests was compiled on the IBM 3083, and all executable tests were linked and executed. Results were printed from the IBM 3083. The tests were reviewed by the validation team and showed acceptable results.

In parallel with the full validation on the IBM 3083, a subset of the ACVC Version 1.8 was executed on an IBM 4341 under UTS, Version 2.3. The subset of sixty tests consisted of 5 tests selected at random from the classes of tests within each chapter. The tests were compiled and executed (when applicable) on the IBM 4341. The test results were transferred from the IBM 4341 to the IBM 3083 via a Remote Spooling Communications Subsystem (RSCS) and printed from the IBM 3083. The tests were reviewed by the validation team and showed acceptable results.

The compiler was tested using command scripts provided by Intermetrics Inc. and reviewed by the validation team. The following options were in effect for testing:

Option	Effect
-quiet	gives a sparse listing
-opt cg assembler=false	don't produce assembler listing

Tests were compiled, linked, and executed (as appropriate) using a single host and target computer. Test output, compilation listings, and job logs were captured on magnetic tape and archived at the AVF. The listings examined on-site by the validation team were also archived.

3.7.3 Test Site

The validation team arrived at Intermetrics Inc., Cambridge MA on 22 September 1986 and departed after testing was completed on 26 September 1986.

APPENDIX A

COMPLIANCE STATEMENT

Intermetrics Inc. has submitted the following compliance statement concerning the Intermetrics 370/UTS Ada Compiler.

COMPLIANCE STATEMENT

Compliance Statement

Configuration:

Compiler: Intermetrics 370/UTS Ada® Compiler, Version 201.16c

Test Suite: Ada Compiler Validation Capability, Version 1.8

Host and Target Computer:

Machine:	IBM 3083
Operating System:	UTS, Version 2.3

Host and Target Computer:

Machine:	IBM 4341
Operating System:	UTS, Version 2.3

Intermetrics Inc. has made no deliberate extensions to the Ada language standard.

Intermetrics Inc. agrees to the public disclosure of this report.

Intermetrics Inc. agrees to comply with the Ada trademark policy, as defined by the Ada Joint Program Office.

Date: 9/22/86

Intermetrics Inc.
Dennis D. Struble
Manager, Ada Compilers

®Ada is a registered trademark of the United States Government (Ada Joint Program Office).

APPENDIX B

APPENDIX F OF THE Ada STANDARD

The only allowed implementation dependencies correspond to implementation-dependent pragmas, to certain machine-dependent conventions as mentioned in chapter 13 of MIL-STD-1815A, and to certain allowed restrictions on representation classes. The implementation-dependent characteristics of the Intermetrics 370/UTS Ada Compiler, Version 201.16c, are described in the following sections which discuss topics in Appendix F of the Ada Language Reference Manual (ANSI/MIL-STD-1815A). The specification of the package STANDARD is also included in this appendix.

```
package STANDARD is

     ...

     type INTEGER is range -2147483648 .. 2147483647;

     type SHORT_FLOAT is digits 6 range -16#0.ffffff#e63 .. 16#0.ffffff#e63;
     type FLOAT is digits 15 range -16#0.fffffffffffff#e63 ..
               16#0.fffffffffffff#e63;

     type DURATION is delta 2.0 ** (-14) range -86400.0 .. 86400.0;
       -- DURATION'SMALL = 2.0 ** (-14)

     ...

end STANDARD;
```

Appendix F. IMPLEMENTATION DEPENDENCIES

This section constitutes Appendix F of the Ada LRM for this implementation. Appendix F from the LRM states:

> *The Ada language allows for certain machine-dependencies in a controlled manner. No machine-dependent syntax or semantic extensions or restrictions are allowed. The only allowed implementation-dependencies correspond to implementation-dependent pragmas and attributes, certain machine-dependent conventions as mentioned in Chapter 13, and certain allowed restrictions on representation clauses.*
>
> *The reference manual of each Ada implementation must include an appendix (called Appendix F) that describes all implementation-dependent characteristics. The Appendix F for a given implementation must list in particular:*
>
> 1. *The form, allowed places, and effect of every implementation-dependent pragma.*
> 2. *The name and the type of every implementation-dependent attribute.*
> 3. *The specification of the package SYSTEM (see 13.7).*
> 4. *The list of all restrictions on representation clauses (see 13.1).*
> 5. *The conventions used for any implementation-generated name denoting implementation-dependent components (see 13.4).*
> 6. *The interpretation of expressions that appear in address clauses, including those for interrupts (see 13.5).*
> 7. *Any restriction on unchecked conversions (see 13.10.2).*
> 8. *Any implementation-dependent characteristics of the input-output packages (see 14).*

In addition, the present section will describe the following topics:

9. Any implementation-dependent rules for termination of tasks dependent on library packages (see 9.4:13).
10. Other implementation dependencies.
11. Compiler capacity limitations.

F.1 Pragmas

This section describes the form, allowed places, and effect of every implementation-dependent pragma.

F.1.1 Pragmas LIST, OPTIMIZE, PAGE, PRIORITY

Pragmas LIST, OPTIMIZE, and PAGE are ignored. Pragma PRIORITY is supported exactly in the form, in the allowed places, and with the effect as described in the LRM.

F.1.2 Pragma SUPPRESS

Form: As specified in LRM B(14) : SUPPRESS

Allowed Places: As specified in LRM B(14) : SUPPRESS

Effect: Pragma SUPPRESS is ignored.

F.1.3 Pragma INLINE

Form: Pragma INLINE (SubprogramNameCommaList)

Allowed Places: As specified in LRM B(4) : INLINE

Effect: If the subprogram body is available, and the subprogram is not recursive, the code is expanded in-line at every call site and is subject to all optimizations.

The stack-frame needed for the elaboration of the inline subprogram will be allocated as a temporary in the frame of the containing code.

Parameters will be passed properly, by value or by reference, as for non-inline subprograms. Register-saving and the like will be suppressed. Parameters may be stored in the local stack-frame or held in registers, as global code generation allows.

Exception-handlers for the INLINE subprogram will be handled as for block-statements.

Use: This pragma is used either when it is believed that the time required for a call to the specified routine will in general be excessive (this for frequently called subprograms) or when the average expected size of expanded code is thought to be comparable to that of a call.

F.1.4 Pragma INTERFACE

Form: Pragma INTERFACE (language_name, subprogram_name)
where the language_name must be an enumeration value of the type

SYSTEM.Supported_Language_Name (see Package SYSTEM below).

Allowed Place: As specified in LRM B(5) : INTERFACE.

Effect: Specifies that a subprogram will be provided outside the Ada program library and will be callable with a specified calling interface. Neither an Ada body nor an Ada body_stub may be provided for a subprogram for which INTERFACE has been specified.

Use: Use with a subprogram being provided via another programming language and for which no body will be given in any Ada program. See also the LINK_NAME pragma.

The calling conventions for an Ada program calling a non-Ada subprogram are described in the Run-Time Model B-5.

F.1.5 Pragma LINK_NAME

Form: Pragma LINK_NAME (subprogram_name, link_name)

Allowed Places: As specified in LRM B(5) for pragma INTERFACE.

Effect: Associates with subprogram subprogram_name the name link_name as its entry point name.

Use: To allow Ada programs, with help from INTERFACE pragma, to reference non-Ada subprograms. Also allows non-Ada programs to call specified Ada subprograms.

F.1.6 Pragma CONTROLLED

Form: Pragma CONTROLLED (AccessTypeName)

Allowed Places: As specified in LRM B(2) : CONTROLLED.

Effect: Ensures that heap objects are not automatically reclaimed. Since no automatic garbage collection is provided, this pragma currently has no effect.

F.1.7 Pragma PACK

Form: Pragma PACK (type_simple_name)

Allowed Place: As specified in LRM 13.1(12)

Effect: Components are allowed their minimal number of storage units as provided for by their own representation and/or packing.

Floating-point components are aligned on storage-unit boundaries. either 4 bytes or 8 bytes, depending on digits.

Use: Pragma PACK is used to reduce storage size. This can allow records and arrays, in some cases, to be passed by value instead of by reference.

Size reduction usually implies an increased cost of accessing components. The decrease in storage size may be offset by increase in size of accessing code and by slowing of accessing operations.

F.1.8 Pragmas SYSTEM_NAME, STORAGE_UNIT, MEMORY_SIZE

These pragmas are not supported and are ignored.

F.2 Implementation-dependent Attributes

This section describes the name and the type of every implementation-dependent attribute.

There are no implementation defined attributes. These are the values for certain language-defined, implementation-dependent attributes:

```
Type INTEGER.
     INTEGER'SIZE                    = 32 -- bits.
     INTEGER'FIRST                   = - (2**31)
     INTEGER'LAST                    =  (2**31-1)


Type SHORT_FLOAT.
     SHORT_FLOAT'SIZE                = 32 -- bits.
     SHORT_FLOAT'DIGITS              = 6
     SHORT_FLOAT'MANTISSA            = 21
     SHORT_FLOAT'EMAX                = 84
     SHORT_FLOAT'EPSILON             = 2.0**(-20)
     SHORT_FLOAT'SMALL               = 2.0**(-85)
     SHORT_FLOAT'LARGE               = 2.0**84
     SHORT_FLOAT'MACHINE_ROUNDS      = false
     SHORT_FLOAT'MACHINE_RADIX       = 16
     SHORT_FLOAT'MACHINE_MANTISSA    = 6
     SHORT_FLOAT'MACHINE_EMAX        = 63
     SHORT_FLOAT'MACHINE_EMIN        =-64
     SHORT_FLOAT'MACHINE_OVERFLOWS   = false
     SHORT_FLOAT'SAFE_EMAX           = 252
     SHORT_FLOAT'SAFE_SMALL          = 16#0.800000#E-63
     SHORT_FLOAT'SAFE_LARGE          = 16#0.FFFFF8#E63

Type FLOAT.
     FLOAT'SIZE                      = 64 -- bits.
     FLOAT'DIGITS                    = 15
     FLOAT'MANTISSA                  = 51
     FLOAT'EMAX                      = 204
     FLOAT'EPSILON                   = 2.0**(-50)
     FLOAT'SMALL                     = 2.0**(-205)
     FLOAT'LARGE                     = (1.0-2**(-51))*2.0**204
     FLOAT'MACHINE_ROUNDS            = false
     FLOAT'MACHINE_RADIX             = 16
     FLOAT'MACHINE_MANTISSA          = 14
     FLOAT'MACHINE_EMAX              = 63
     FLOAT'MACHINE_EMIN              = -64
     FLOAT'MACHINE_OVERFLOWS         = false
```

```
    FLOAT'SAFE_EMAX                 = 252
    FLOAT'SAFE_SMALL                = 16#0.80000000000000#E-63
    FLOAT'SAFE_LARGE                = 16#0.FFFFFFFFFFFFE0#E63

Type DURATION.
    DURATION'DELTA                  =  2.0**(-14) -- seconds
    DURATION'FIRST                  = - 86,400
    DURATION'LAST                   =  86,400
    DURATION'SMALL                  =  2.0**(-14)

Type PRIORITY.
    PRIORITY'FIRST                  = -128
    PRIORITY'LAST                   =  127
```

F.3 Package SYSTEM

```
package SYSTEM is

type ADDRESS is private;    -- "=", "/=" defined implicitly;
type NAME is (UTS, MVS, CMS, Prime50, Sperry1100,
              MIL_STD_1750A);

SYSTEM_NAME : constant NAME := UTS;

STORAGE_UNIT : constant := 8;
MEMORY_SIZE : constant := 2**24;  -- 2**31 for XA mode
        -- In storage units

        -- System-Dependent Named Numbers:

MIN_INT : constant := INTEGER'POS(INTEGER'FIRST);
MAX_INT : constant := INTEGER'POS(INTEGER'LAST);
MAX_DIGITS : constant := 15;
MAX_MANTISSA : constant := 31;
FINE_DELTA : constant := 2.0**(-31);
TICK : constant := 1.0;
        -- Minimum process delay is 1.0 second on UTS
        -- although clock can resolve to 0.001 second.

        -- Other System-Dependent Declarations

subtype PRIORITY is INTEGER range -127..127;

-----------------------------------------------------
-- Implementation-dependent additions to package SYSTEM --
-----------------------------------------------------

NULL_ADDRESS : constant ADDRESS;
        -- Same bit pattern as "null" access value
        -- This is the value of 'ADDRESS for named numbers
        -- The 'ADDRESS of any object which occupies storage
        --  is NOT equal to this value.

ADDRESS_SIZE : constant := 32;
        -- Number of bits in ADDRESS objects.
        --          = ADDRESS'SIZE, but static.

ADDRESS_SEGMENT_SIZE : constant := 2**24;
        -- Number of storage units in address segment
```

```
type ADDRESS_OFFSET is new INTEGER;
        -- Used for address arithmetic
type ADDRESS_SEGMENT is new INTEGER;
        -- Always zero on targets with
        --     unsegmented address space.

subtype NORMALIZED_ADDRESS_OFFSET is
  ADDRESS_OFFSET range 0 .. ADDRESS_SEGMENT_SIZE - 1;
        -- Range of address offsets returned by OFFSET_OF

function "+"(addr : ADDRESS; offset : ADDRESS_OFFSET)
                    return ADDRESS;
function "+"(offset : ADDRESS_OFFSET; addr : ADDRESS)
                    return ADDRESS;
        -- Provide addition between addresses and
        --  offsets.  May cross segment boundaries on targets
        --  where objects may span segments.
        -- On other targets, CONSTRAINT_ERROR will be raised
        --  when OFFSET_OF(addr) + offset not in
        --  NORMALIZED_ADDRESS_OFFSET.

function "-"(left, right : ADDRESS) return ADDRESS_OFFSET;
        -- May exceed SEGMENT_SIZE on targets where objects
        --  may span segments.
        -- On other targets, CONSTRAINT_ERROR
        --  will be raised if
        --            SEGMENT_OF(left) /= SEGMENT_OF(right).

function "-"(addr : ADDRESS; offset : ADDRESS_OFFSET) return
  ADDRESS;
        -- Provide subtraction of addresses and offsets.
        -- May cross segment boundaries on targets where
        --  objects may span segments.
        -- On other targets, CONSTRAINT_ERROR will be raised when
        --  (OFFSET_OF(addr) - offset)
        --   not in NORMALIZED_ADDRESS_OFFSET.

function OFFSET_OF (addr : ADDRESS)
                    return NORMALIZED_ADDRESS_OFFSET;
        -- Extract offset part of ADDRESS
        -- Always in range 0..seg_size - 1

function SEGMENT_OF (addr : ADDRESS) return ADDRESS_SEGMENT;
        -- Extract segment part of ADDRESS
        -- (zero on targets with unsegmented address space)
```

```
    function MAKE_ADDRESS (offset : ADDRESS_OFFSET;
                           segment : ADDRESS_SEGMENT := 0)
                                return ADDRESS;
            -- Build address given an offset and a segment.
            -- Offset may be > seg_size on targets where objects
            --  may span segments, in which case it is equiv
            --  to "MAKE_ADDRESS(0,segment) + offset".
            -- On other targets, CONSTRAINT_ERROR will be raised
    --  when offset not in NORMALIZED_ADDRESS_OFFSET.

    type Supported_Language_Name is (   -- Target dependent
            -- The following are "foreign" languages:
       ASSEMBLER,
       FORTRAN_MAIN,
       FORTRAN,
       COBOL_MAIN,
       COBOL,
       JOVIAL_MAIN,
       PL1_MAIN,

       AIE_ASSEMBLER, -- NOT a "foreign" language - uses AIE RTS
       UNSPECIFIED_LANGUAGE_MAIN,
       UNSPECIFIED_LANGUAGE
    );
            -- Most/least accurate built-in integer and float types

    subtype LONGEST_INTEGER is STANDARD.INTEGER;
    subtype SHORTEST_INTEGER is STANDARD.INTEGER;

    subtype LONGEST_FLOAT is STANDARD.FLOAT;
    subtype SHORTEST_FLOAT is STANDARD.SHORT_FLOAT;

private

    type ADDRESS is access INTEGER;
            -- Note: The designated type here (INTEGER) is
            --       irrelevant.  ADDRESS is made an access type
            --       simply to guarantee it has the same size as
            --       access values, which are single addresses.
            --       Allocators of type ADDRESS are NOT meaningful.

    NULL_ADDRESS : constant ADDRESS := null;

end SYSTEM ;
```

F.4 Representation Clauses

This section describes the list of all restrictions on representation clauses.

"*NOTE: An implementation may limit its acceptance of representation clauses to those that can be handled simply by the underlying hardware.... If a program contains a representation clause that is not accepted [by the compiler], then the program is illegal.*" (LRM 13.1(10)).

There are no restrictions except as follows:

a. Length clauses are not allowed.

b. Representation clauses for enumeration types are not allowed.

c. Address clauses are not allowed.

d. Record-representation-clause:

   Within a record-representation-clause, the object being represented must be no larger than one 32-bit word.

   The range of bits specified must be in the range of 0..31.

   Record components, including those generated implicitly by the compiler, whose locations are not given by the representation-clause, are layed out by the compiler following all the components whose locations are given by the representation-clause. Such components of the invariant part of the record are allocated to follow the user-specified components of the invariant part, and such components in any given variant part are allocated to follow the user-specified components of that variant part.

F.5 Implementation-dependent Components

This section describes the conventions used for any implementation-generated name denoting implementation-dependent components.

There are no implementation-generated names denoting implementation-dependent (record) components, although there are, indeed, such components. Hence, there is no convention (or possibility) of naming them and, therefore, no way to offer a representation clause for such components.

NOTE: Records containing dynamic-sized components will contain (generally) unnamed offset components which will "point" to the dynamic-sized components stored later in the record. ACS 370/UTS offers no means to specify the representation of such components.

F.6 Address Clauses

This section describes the interpretation of expressions that appear in address clauses, including those for interrupts.

Address clauses are not allowed.

F.7 Unchecked Conversions

This section describes any restrictions on unchecked conversions.

The source and target values must both be of an integer, enumeration, or access type.

F.8 Input-Output

This section describes implementation-dependent characteristics of the input-output packages.

(a) Declaration of type Direct_IO.Count? [14.2.5]
0..Integer'last;

(b) Effect of input/output for access types?
Not meaningful if read by different program invocations

(c) Disposition of unclosed IN_FILE files at program termination? [14.1(7)]
Files are closed.

(d) Disposition of unclosed OUT_FILE files at program termination? [14.1(7)]
Files are closed.

(e) Disposition of unclosed INOUT_FILE files at program termination? [14.1(7)]
Files are closed.

(f) Form of, and restrictions on, file names? [14.1(1)]
UTS filenames

(g) Possible uses of FORM parameter in I/O subprograms? [14.1(1)]
The image of an integer specifying the UTS file protection on CREATE.

(h) Where are I/O exceptions raised beyond what is described in Chapter 14? [14.1(11)]
None raised.

(i) Are alternate specifications (such as abbreviations) allowed for file names? If so, what is the form of these alternatives? [14.2.1(21)]
No.

(j) When is DATA_ERROR *not* raised for sequential or direct input of an inappropriate ELEMENT_TYPE? [14.2.2(4), 14.2.4(4)]
When it can be assigned without CONSTRAINT_ERROR to a variable of ELEMENT_TYPE.

(k) What are the standard input and standard output files? [14.3(5)]
UTS standard input and output

(l) What are the forms of line terminators and page terminators? [14.3(7)]
Line terminator is ASCII.LF (line feed);
page terminator is ASCII.FF (form feed)

(m) Value of Text_IO.Count'last? [14.3(8)]
integer'last

(n) Value of Text_IO.Field'last? [14.3.7(2)]
integer'last

(o) Effect of instantiating ENUMERATION_IO for an integer type? [14.3.9(15)]

The instantiated Put will work properly, but the instantiated Get will raise Data_Error

(p) Restrictions on types that can be instantiated for input/output?

Neither direct I/O nor sequential I/O can be instantiated for an unconstrained array type or for an unconstrained record type lacking default values for its discriminants.

(q) Specification of package Low_Level_IO? [14.6]

Low_Level_IO is not provided.

F.9 Tasking

This section describes implementation-dependent characteristics of the tasking run-time packages.

Even though a main program completes and terminates (its dependent tasks, if any, having terminated), the elaboration of the program as a whole continues until each task dependent upon a library unit package has either terminated or reached an open terminate alternative. See LRM 9.4(13).

F.10 Other Matters

This section describes other implementation-dependent characteristics of the system.

a. Restrictions on SHARED variables (LRM 9.11):

Must be of a scalar or access type.

b. Package Machine_Code

Will not be provided.

c. Order of compilation of generic bodies and subunits (LRM 10.3:9):

Body and subunits of generic must be in the same compilation as the specification if instantiations precede them (see AI-00257/02).

F.11 Compiler Limitations

(a) Maximum length of source line?
255 characters.

(b) Maximum number of "use" scopes?
Limit is 50, set arbitrarily by SEMANTICS as maximum number of distinct packages actively "used."

(c) Maximum length of identifier?
255 characters.

(d) Maximum number of nested loops?
24 nested loops.

APPENDIX C

TEST PARAMETERS

Certain tests in the ACVC make use of implementation-dependent values, such as the maximum length of an input line and invalid file names. A test that makes use of such values is identified by the extension .TST in its file name. Actual values to be substituted are represented by names that begin with a dollar sign. A value must be substituted for each of these names before the test is run. The values used for this validation are given below.

Name and Meaning	Value
$BIG_ID1 Identifier the size of the maximum input line length with varying last character.	(1..254 =>'A', 255 =>'1')
$BIG_ID2 Identifier the size of the maximum input line length with varying last character.	(1..254 =>'A', 255 =>'2')
$BIG_ID3 Identifier the size of the maximum input line length with varying middle character.	(1..127 =>'A', 128 =>'3', 129..255 =>'A')
$BIG_ID4 Identifier the size of the maximum input line length with varying middle character.	(1..127 =>'A', 128 =>'4', 129..255 =>'A')
$BIG_INT_LIT An integer literal of value 298 with enough leading zeroes so that it is the size of the maximum line length.	(1..252 =>'0', 253..255 =>"298")

TEST PARAMETERS

Name and Meaning	Value
$BIG_REAL_LIT A real literal that can be either of floating- or fixed-point type, has value 690.0, and has enough leading zeroes to be the size of the maximum line length.	(1..249 =>'0', 250..255 =>"69.0E1")
$BLANKS A sequence of blanks twenty characters fewer than the size of the maximum line length.	(1..235 =>' ')
$COUNT_LAST A universal integer literal whose value is TEXT_IO.COUNT'LAST.	2_147_483_647
$EXTENDED_ASCII_CHARS A string literal containing all the ASCII characters with printable graphics that are not in the basic 55 Ada character set.	"abcdefghijklmnopqrstuvwxyz!$%?@[\]^`{}~"
$FIELD_LAST A universal integer literal whose value is TEXT_IO. FIELD'LAST.	2_147_483_647
$FILE_NAME_WITH_BAD_CHARS An illegal external file name that either contains invalid characters, or is too long if no invalid characters exist.	X}]!@/#$^-Y
$FILE_NAME_WITH_WILD_CARD_CHAR An external file name that either contains a wild card character, or is too long if no wild card character exists.	WILDCARDS/DONT/MATTER
$GREATER_THAN_DURATION A universal real value that lies between DURATION'BASE'LAST and DURATION'LAST if any, otherwise any value in the range of DURATION.	90_000.0
$GREATER_THAN_DURATION_BASE_LAST The universal real value that is greater than DURATION'BASE'LAST, if such a value exists.	10_000_000.0

Name and Meaning	Value
$ILLEGAL_EXTERNAL_FILE_NAME1 An illegal external file name.	BAD-CHARAC/TER*^
$ILLEGAL_EXTERNAL_FILE_NAME2 An illegal external file name that is different from $ILLEGAL_EXTERNAL_FILE_NAME1.	NO/MUCH-TOO-LONG-NAME-FOR-A-FILE
$INTEGER_FIRST The universal integer literal expression whose value is INTEGER'FIRST.	-2_147_483_648
$INTEGER_LAST The universal integer literal expression whose value is INTEGER'LAST.	2_147_483_647
$LESS_THAN_DURATION A universal real value that lies between DURATION'BASE'FIRST and DURATION'FIRST if any, otherwise any value in the range of DURATION.	-90_000.0
$LESS_THAN_DURATION_BASE_FIRST The universal real value that is less than DURATION'BASE'FIRST, if such a value exists.	-10_000_000.0
$MAX_DIGITS The universal integer literal whose value is the maximum digits supported for floating-point types.	15
$MAX_IN_LEN The universal integer literal whose value is the maximum input line length permitted by the implementation.	255
$MAX_INT The universal integer literal whose value is SYSTEM.MAX_INT.	2_147_483_647

TEST PARAMETERS

Name and Meaning	Value
$NAME A name of a predefined numeric type other than FLOAT, INTEGER, SHORT_FLOAT, SHORT_INTEGER, LONG_FLOAT, or LONG_INTEGER if one exists, otherwise any undefined name.	NO_OTHER_PREDEF_NUM_TYPE
$NEG_BASED_INT A based integer literal whose highest order nonzero bit falls in the sign bit position of the representation for SYSTEM.MAX_INT.	16#FFFFFFFE#
$NON_ASCII_CHAR_TYPE An enumerated type definition for a character type whose literals are the identifier NON_NULL and all non-ASCII characters with printable graphics.	(NON_NULL)

APPENDIX D

WITHDRAWN TESTS

Some tests are withdrawn from the ACVC because they do not conform to the Ada Standard. The following 19 tests had been withdrawn at the time of validation testing for the reasons indicated. A reference of the form "AI-ddddd" is to an Ada Commentary.

- C32114A: An unterminated string literal occurs at line 62.
- B33203C: The reserved word "IS" is misspelled at line 45.
- C34018A: The call of function G at line 114 is ambiguous in the presence of implicit conversions.
- C35904A: The elaboration of subtype declarations SFX3 and SFX4 may raise NUMERIC_ERROR instead of CONSTRAINT_ERROR as expected in the test.
- B37401A: The object declarations at lines 126 through 135 follow subprogram bodies declared in the same declarative part.
- C41404A: The values of 'LAST and 'LENGTH are incorrect in the if statements from line 74 to the end of the test.
- B45116A: ARRPRIBL1 and ARRPRIBL2 are initialized with a value of the wrong type--PRIBOOL_TYPE instead of ARRPRIBOOL_TYPE--at line 41.
- C48008A: The assumption that evaluation of default initial values occurs when an exception is raised by an allocator is incorrect according to AI-00397.
- B49006A: Object declarations at lines 41 and 50 are terminated incorrectly with colons, and end case; is missing from line 42.
- B4A010C: The object declaration in line 18 follows a subprogram body of the same declarative part.

WITHDRAWN TESTS

- B74101B: The begin at line 9 causes a declarative part to be treated as a sequence of statements.

- C87B50A: The call of "/=" at line 31 requires a use clause for package A.

- C92005A: The "/=" for type PACK.BIG_INT at line 40 is not visible without a use clause for the package PACK.

- C940ACA: The assumption that allocated task TT1 will run prior to the main program, and thus assign SPYNUMB the value checked for by the main program, is erroneous.

- CA3005A..D (4 tests): No valid elaboration order exists for these tests.

- BC3204C: The body of BC3204C0 is missing.

END

6-87

DTIC